with different scalability tags are available to the projector module **60**. At runtime, the projector module **60** selects the objects with the appropriate scalability tags.

For example, if the movie is distributed via a CD with a CD build file including all of the objects created for the movie, the projector module **60** scales the 3D graphics based on the detected CPU speed. If the host computer **55** includes a Pentium **166** processor, the projector module **60** selects the objects tagged with a Base PC tag **200***c*. If the same CD is played on a host computer **55** with a Pentium II processor, the graphics displayed scales to the higher CPU speed and includes additional graphics and/or higher resolution objects tagged with a PII/266 tag **200***d*.

Normally, the runtime engine will load and play data tagged with scalability flags that reasonably match the host computer's configuration. However, when doing an Internet build, extra sets of scalability data may be culled out to reduce the size of the download. For example a low bandwidth Internet build might only include data tagged as Base PC and Software rendering. At playback time the projector module **60** detects, from extra header information in the build, that the build only includes a limited set of scalability data, and allows playback of this data even though it may not match the current machine configuration.

F. DYNAMIC REPLACEMENT OF 3D OBJECTS IN A 3D PROJECT LIBRARY.

The system's library structure as described above in conjunction with FIGS. **2–4** allows a 3D object to be defined as a library object and its geometry information be stored outside a scene. During the playing of a scene, the projector module **30** examines each node's name and inquires if the name corresponds to a root-node listed in the scene's 3DS file. If the answer is YES, the node belongs to a library object. The projector module **30** then retrieves the corresponding master library object file and compares the nodes following the root-node in the scene for a match in the master library object file. If a match is found, the geometry data in the master library object file is used for that node. Accordingly, updates to a 3D object may be made by changing the object's geometry in the library file instead of making the change in each scene in which the object appears.

According to one embodiment of the invention, the 3D movie incorporates commercial products that are replaced and updated based on the sponsorship available for the product. For instance, the Coca-Cola company may pay advertisement fees to have one of its products displayed in one or more scenes of the movie. If, after inserting the Coca-Cola product into the scenes, the Coca-Cola product is to be replaced with a Pepsi-Cola product, the geometry data for the object is changed in the master library object file without having to manually make the changes in each scene that the object appears. Thus, changes to the movie based on sponsorship changes may be efficiently reflected through the system's library substitution method.

The present library substitution method, in conjunction with the system's method of file organization for Internet streaming, also facilitates the transmission of updated versions of movies over the Internet. As described above in conjunction with FIGS. **16–18**, the publishing module **30** packages data to be delivered over the Internet into the upfront file and one or more streaming files. The geometry information associated with a 3D object is stored in the upfront file. Thus, if a change to a 3D object's geometry is made in a movie that has already been downloaded by the projector module **60**, the projector module **60** needs not to download the entire movie again. Instead, the projector module **60** just downloads the upfront file with the new geometry information. As described above, the projector module **60** detects a change in the downloaded upfront file by checking the file's checksum number.

G. CONTROLLING 3D OBJECT GESTURES

The present system's library structure as described above in conjunction with FIGS. **2–4** also allows gesture animations to be defined within a master library object file. The gesture animations are overlaid on top of a scene's animation when triggered within the scene.

FIG. **17** is a flow process diagram of a software program for processing gesture animations for a scene. The program starts, and in step **280**, inquires if there are any more messages to process in the scene's message file. If the answer is YES, the program asks in step **282** if the message is a gesture message. If the answer is again YES, the program, in step **284**, searches the corresponding master library object file for the gesture tag corresponding to the gesture.

In step **286**, the program retrieves the keyframes associated with the gesture from the actor's S3D file, and in step **288**, merges the retrieved keyframes with the keyframes created for the scene in the scene's S3D file.

One type of gesture used in the system are lip-synch gestures. FIG. **18** is a flow diagram of a software program for creating audio and lip animation data for an actor for lip-synching purposes. The program starts, and in step **300**, it records generic phonemes for an actor. In step **302**, the program stores the phonemes into -he sound subdirectory **207** of the project directory **205** (FIG. **5**). In step **304**, the animator uses 3D Studio MAX to create lip gestures to match the recorded phonemes.

Once the phonemes and corresponding lip gestures have been created for the actor, the program, in step **306**, records a narration for the actor according to the movie script. The audio recording is also stored in the sound subdirectory **207** as an audio file with a ".wav" extension. In step **308**, the program performs a lip-synch analysis of the audio file according to conventional methods described, for instance, in Juang et. al, *Fundamentals of Speech Recognition* (Prentice Hall 1993), which is incorporated herein by reference. The program further creates a series of lip-synch messages for the actor with gesture tags that correspond to the phonemes in the audio file. The lip-synch gesture messages are stored in the sound subdirectory **207** as a lip-synch file with a ".sync" extension. The lip-synch gestures in the lip-synch file are triggered at run-time and overlaid on top of a current scene's animation.

What is claimed is:

1. A method for creating 3D animated content for multiple target machines from a single production process, the animated content including a plurality of scenes, each scene including a 3D object having a plurality of nodes, each node identifying a discrete piece of 3D geometry making up the 3D object, the method comprising:

creating a first version of a node of the 3D object identifying a first piece of 3D geometry data;

creating a second version of the node identifying a second piece of 3D geometry data;

tagging each version of the node with a tag identifier, the tag identifier identifying each version of the node as suitable for display in a particular type of machine;

storing the first and second versions of the node in a library model file, the library model file residing in a